**Оптимизация PHP кода**

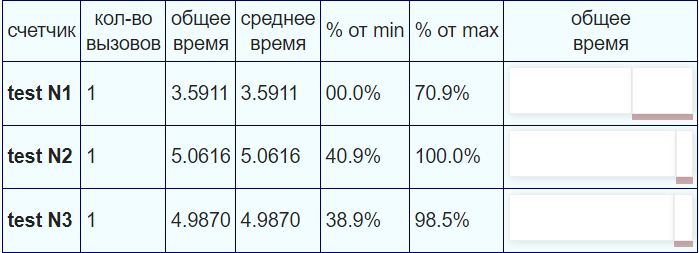
Способы оптимизировать любую (готовую) программу, не меняя ни одного алгоритма:

### **Выносите $переменные из "текстовых строк" - ускорение 25-40%**

Одна и та же операция присваивания (либо echo/print для вывода на экран) в зависимости от того, заключены ли переменные в кавычки или нет, сильно влияет на скорость. В первом и втором вариантах добавлены пробелы, чтобы выровнять размер общего кода для парсинга.

1. **{$x="test".$test; }**
2. **{$x="test $test"; }**
3. **{$x="test";$x.=$test;}**

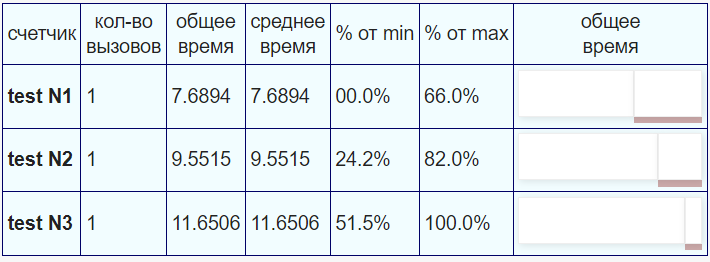
Переменная $test содержит строку "1234567890".



Итак, никогда не пишите $a="$b", ибо это затормозит программу (в этой строке) на 40%.

Однако, если у вас большая строка, где много текста и переменных, различия в скорости уменьшаются, т.к. общие затраты на парсинг становятся намного больше, чем разные по эффективности команды. Но почему бы и не увеличить скорость программы (строк присваивания) почти на четверть таким простым методом?

1. **{$x="test ".$test." test ".$test." test ".$test; }**
2. **{$x="test $test test $test test $test"; }**
3. **{$x="test ";$x.=$test;$x="test ";$x.=$test;$x="test ";$x.=$test;}**



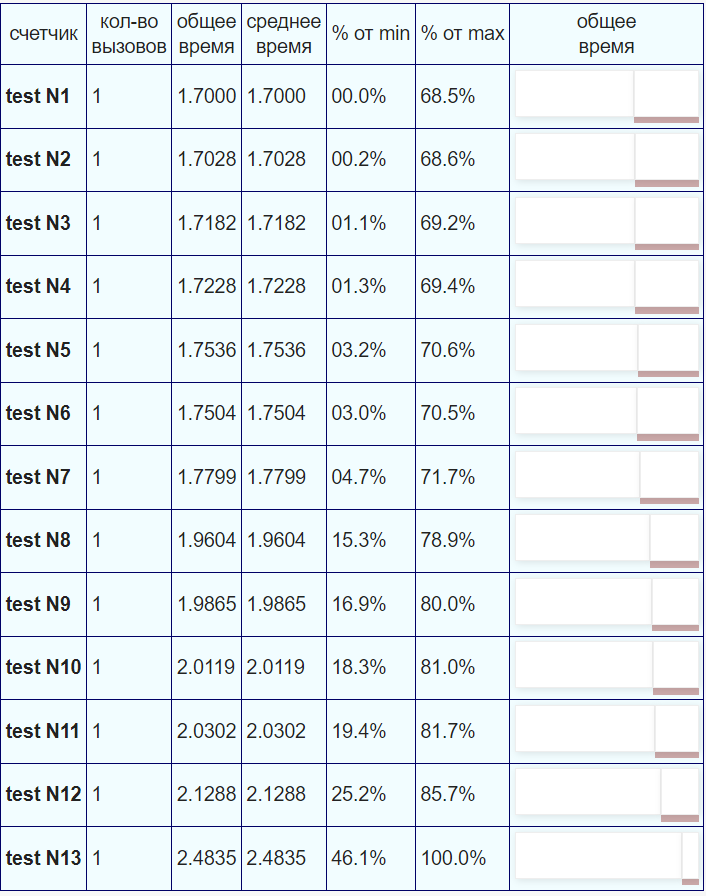
Использование одинарных кавычек ('') выполняется быстрее, чем использование двойных кавычек (""), если вы хотите сохранить только строку внутри нее, избегая любых переменных. Двойные кавычки проверяют наличие переменной и добавляют немного накладных расходов.

### **2. Короткие переменные не более 7 символов - ускорение 15%**

Как влияет длина имен переменных на скорость программы? Если использовать очень длинные переменные - очевидно, что весьма сильно. Однако и с короткими именами не все просто:

1. **{$x=1;}**
2. **{$x2=1;}**
3. **{$x03=1;}**
4. **{$x004=1;}**
5. **{$x0005=1;}**
6. **{$x00006=1;}**
7. **{$x000007=1;}**
8. **{$x0000008=1;}**
9. **{$x000000010=1;}**
10. **{$x00000000012=1;}**
11. **{$x0000000000014=1;}**
12. **{$x000000000000016=1;}**
13. **{$x0000000000000000000000000000032=1;}**

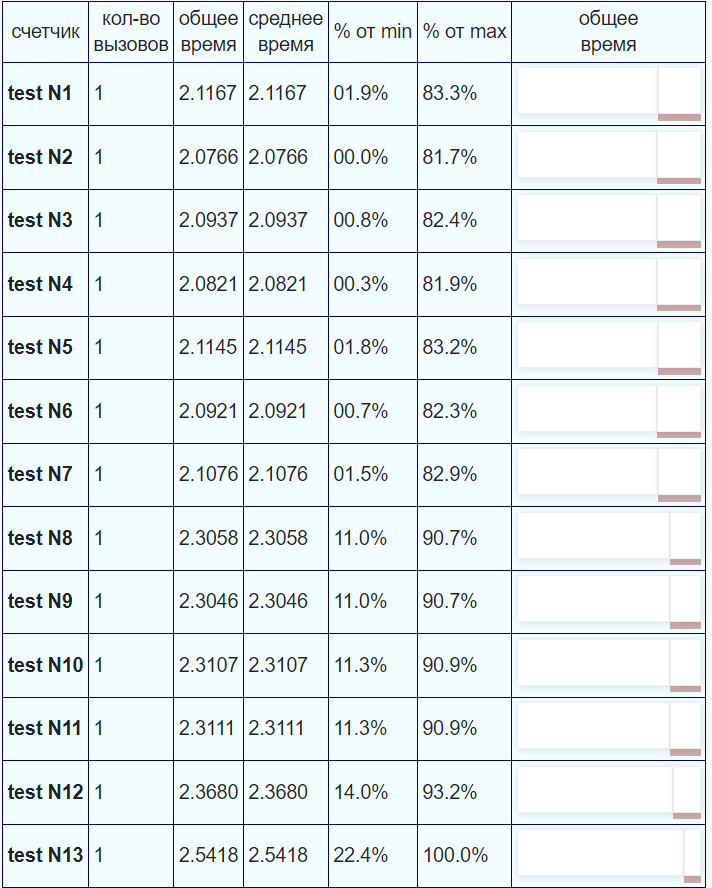
выдает предсказуемый результат:



Переменные от 32 символов могут тормознуть программу почти на половину.

Но если заполнять пробелами (" "), чтобы все строки "$x=1; ..." по длине занимали одно и тоже расстояние, то получается вот что:

1. **{$x=1; }**
2. **{$x2=1; }**
3. **{$x03=1; }**
4. **{$x004=1; }**
5. **{$x0005=1; }**
6. **{$x00006=1; }**
7. **{$x000007=1; }**
8. **{$x0000008=1; }**
9. **{$x000000010=1; }**
10. **{$x00000000012=1; }**
11. **{$x0000000000014=1; }**
12. **{$x000000000000016=1; }**
13. **{$x0000000000000000000000000000032=1; }**



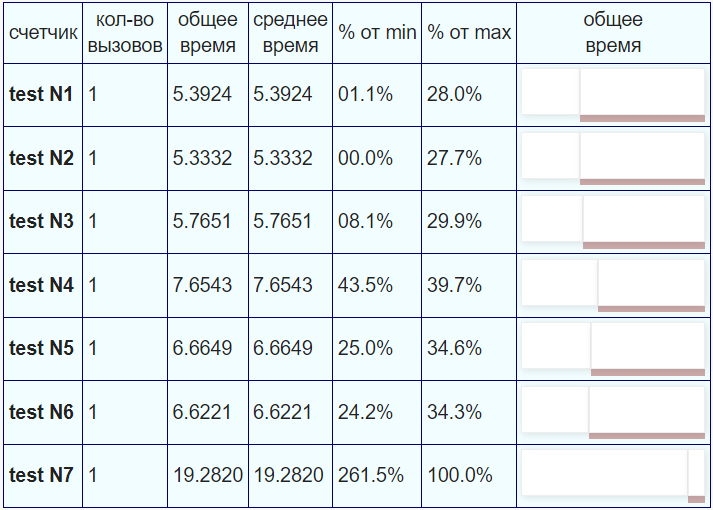
При длине переменных в 8 и более символов происходит резкое снижение производительности, до 15%! А команд, включающих названия переменных, очень много. Еще один менее резкий скачек на переменных с именем 16 символов в длину и более. А в остальных случаях - чем больше, тем дольше, весьма линейная зависимость.

Вывод - не используйте переменные из 8 и более символов, выиграете 15% скорости (вернее, сэкономите).

### **3. Тормозят ли массивы в PHP? Вернее, как именно. Ускорение 40%.**

А вот и не тормозят. Конечно, тест простой, но большой разницы между непрерывным простым (1), простым (2) и ассоциативным (3) массивами нет (элемент 0000 преобразуется в 0 - это же число, а не строка). И уж явно не тормозят "не ассоциативные не сплошные массивы".

1. **{$test[0000]=1;$test[0001]=1;$test[0002]=1;$test[0003]=1;$test[0004]=1; }**
2. **{$test[1000]=1;$test[1001]=1;$test[1002]=1;$test[1003]=1;$test[1004]=1; }**
3. **{$test["aa"]=1;$test["bb"]=1;$test["cc"]=1;$test["dd"]=1;$test["ee"]=1; }**
4. **{$test[aa]=1; $test[bb]=1; $test[cc]=1; $test[dd]=1; $test[ee]=1; }**
5. **{$test[0][0]=1;$test[0][1]=1;$test[0][2]=1;$test[0][3]=1;$test[0][4]=1; }**
6. **{$test[2][1]=1;$test[3][8]=1;$test[4][9]=1;$test[33][99]=1;$test[123][99]=1;}**
7. **{$test[a][b]=1;$test[x][y]=1;$test[d][c]=1;$test[a][s]=1;$test[b][n]=1; }**

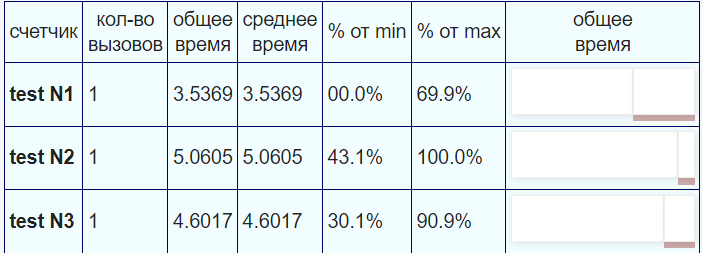


Доступ к элементу одномерного ассоциативного массива по имени, не заключенному в кавычки, тормозит процесс на треть (относительно того же примера, но в кавычках). А вот в двухмерном массиве программа работает медленнее аж в 2.5 раза! После такого теста хочешь не хочешь, а в любой программе пожертвуешь удобством - обращение к элементам массива по имени без кавычек.

### **4. Выносите многомерные массивы из "текстовых строк" - ускорение 25-30%. Одномерные можно не выносить.**

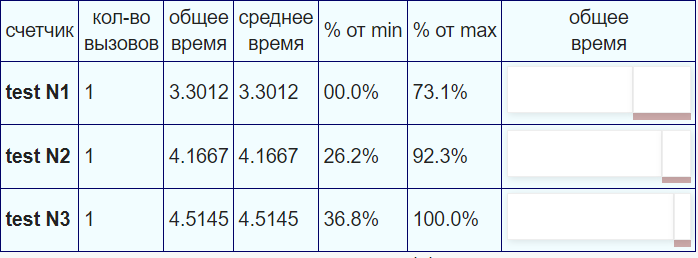
При использовании многомерных массивов в строках наблюдается заметное снижение скорости Из-за многомерности нужно заключать переменные в парные фигурные скобки.

1. **{$x="test ".$myarray["name"]["second"][1]." test"; }**
2. **{$x="test {$myarray[name][second][1]} test"; }**
3. **{$x="test ";$x.=$myarray["name"]["second"][1];$x=" test";}**



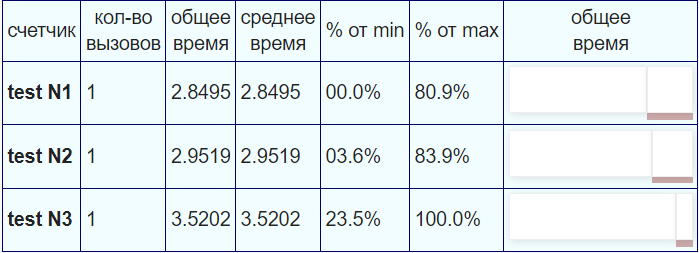
Тот же пример с ассоциативным 3-х мерным массивом, но с обращением к элементу по его индексному номеру:

1. **{$x="test ".$myarray[3][2][0]." test"; }**
2. **{$x="test {$myarray[3][2][0]} test"; }**
3. **{$x="test ";$x.=$myarray[3][2][0];$x=" test";}**



Разница в 1 и 2 вариантах очень мала. Это говорит, что потери на неэффективное использование кавычек не слишком большое, чем доступ к массивам (см. тесты в первой главе).

1. **{$x="test".$myarray["test"]."test";}**
2. **{$x="test$myarray[test]test"; }**
3. **{$x="test{$myarray[test]}test"; }**



А теперь, на основании всех трех тестов, однозначный вывод: использовать фигурные скобки для обозначения границ имени элемента многомерного массива НЕЛЬЗЯ. Это сильно снижает скорость работы - 25-30% (в третьем варианте от простого добавления скобок скорость снизилась на четверть). Не использовать скобки нельзя. Следовательно, единственный способ не терять 30% скорости - выносить многомерные массивы из скобок.

Такой же тест, но для одномерных:

1. **{$x="test".$myarray["test"]."test".$myarray["test"]."test".$myarray["test"]; }**
2. **{$x="test$myarray[test]testtest$myarray[test]testtest$myarray[test]test"; }**
3. **{$x="test{$myarray[test]}testtest{$myarray[test]}testtest{$myarray[test]}test";}**



Сравнивая два последних теста очевидно, что одномерные массивы можно и не выносить, потери всего 3-4% ( а вот на простых переменных - потери 25-40%!).

### 

### 

### **5. Циклы: for, foreach, while, count/sizeof() - ускорение 15%-30%**

В начале программы создается массив $test из целых чисел (100 000 элементов). Потом один раз запускаются приведенные ниже примеры. Цикл проходит данный массив 3-мя способами (разными циклами) и выполняет кое-какие операции. Не выполнять в цикле ничего нельзя, ибо это будет уже совсем не реальный тест.

1. **{$x=0; foreach($test as $n) { $x=sprintf("test%08i",$i); }}**
2. **{$x=0; for ($it=0; $it<100000; $it++) { $x=sprintf("test%08i",$i); }}**
3. **{$x=0; $it=0; while($it<100000) { $x=sprintf("test%08i",$i); $it++; }}**
4. **{$x=0; for ($it=0; $it<count($test); $it++) { $x=sprintf("test%08i",$i); }}**
5. **{$x=0; $it=0; while($it<count($test)) { $x=sprintf("test%08i",$i); $it++; }}**
6. **{$x=0; $co=count($test); for ($it=0; $it<$co; $it++) { $x=sprintf("test%08i",$i); }}**
7. **{$x=0; $co=count($test); $it=0; while($it<$co) { $x=sprintf("test%08i",$i); $it++; }}**



Почему sprintf, а не реальное echo? echo использовать нельзя, т.к. от него будет немерянный буфер (OUTPUT в браузер или консоль).

Теперь о деле. Бесспорный вывод - использование foreach сильно тормозит дело, а между for и while большой разницы нет. (На голом тесте for/while/foreach {..} тормоза foreach - 30%). Это не удивительно, т.к. foreach делает копию массива, на что тратиться масса времени (хотя это только слухи).

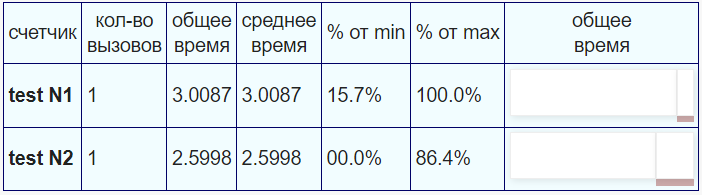
Вывод с count() не столь очевиден, потому что от разного текста в цикле % тормознутости от самого быстрого варианта резко возрастает... Я взял цикл с небольшой нагрузкой - проход по огромному массиву $test + форматирование функцией sprintf. Как видите, варианты с count() и заменяющей эту функцию переменной $co различаются на 10% по скорости между собой (не смотрите на вариант с константой в 100000, заранее знать кол-во элементов невозможно).

Вывод о не ассоциативных массивах: 1) foreach существенно замедляет работу 2) использование count() в простых циклах - замедление 10%. Но на сложных циклах потери от лишних запусков count() будут абсолютно незаметны, так что ситуация не очевидна.

Сравнение count() и sizeof().

Судя по мануалу - это алиасы. Об этом написано на страницах самих функций и дополнительной странице "Appendex => Aliases list". Что же мы видим на массиве в 100000 элементов:

1. **{$x=0; for ($it=0; $it<count($test); $it++) { $x=sprintf("test%08i",$test[$it]);}}**
2. **{$x=0; for ($it=0; $it<sizeof($test); $it++) { $x=sprintf("test%08i",$test[$it]);}}**



Пусть тесты будут иметь погрешности... Но результат один - count() заметно отстает по скорости от sizeof()! Хм, я бы к записи в мануале сделал приписку: "The sizeof() function is an alias for count(), but последний сильно тормозит!"

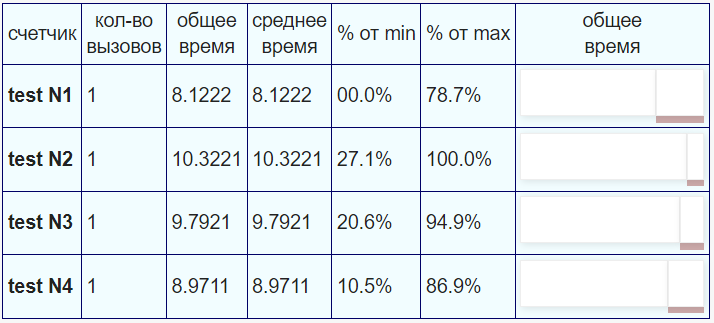
Если кол-во элементов в массиве меньше 65000 (64К), то эти функции по скорости практически не различимы. Тут вывод простой - переходим на использование sizeof(), как ускоренного алиаса count(). Это принесет свои результаты на огромных массивах.

**Ассоциативные массивы: тестирование разных способов перебора**

С ними наблюдается та же проблема: на разных по величине массивах разные функции эффективны, но лучше всех foreach!

Массив в 200 элементов и 1000 повторов программы:

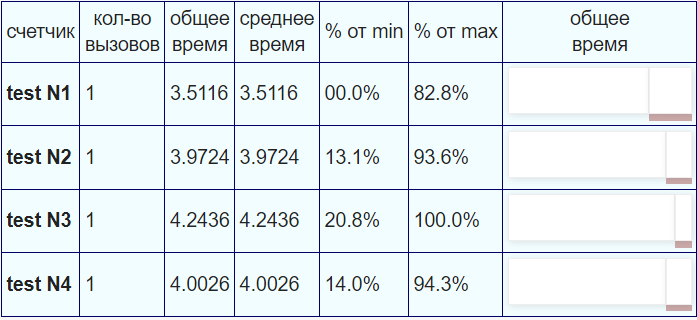
1. **{$x=0; foreach($test as $k=>$v) { $x=sprintf("%s=>%s\n",$k,$v); }}**
2. **{$x=0; reset($test); while (list($k, $v) = each($test)) { $x=sprintf("%s=>%s\n",$k,$v); }}**
3. **{$x=0; $k=array\_keys($test); $co=sizeof($k); for ($it=0; $it<$co; $it++) { $x=sprintf("%s=>%s\n",$k[$it],$test[$k[$it]]); }}**
4. **{$x=0; reset($test); while ($k=key($test)) { $x=sprintf("%s=>%s\n",$k,current($test)); next($test); }}**



Тоже самое, но массив в 5000 элементов и 200 повторов:



Опять тоже самое, но массив в 100 000 элементов и без повторов:



Другие тесты на холостых циклах тоже показывают преимущество foreach.

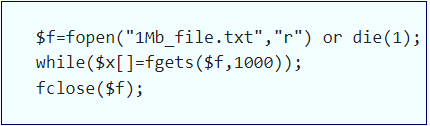
Резюме:

* sizeof() лучше, чем count()
* в циклах sizeof лучше вообще заменить на переменную
* for и while практически не отличимы
* для перебора простых индексных массивов нужно использовать for или while
* для перебора ассоциативных массивов нужно использовать foreach

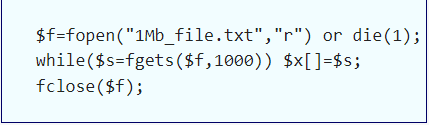
### 

### **7. Для чтения файла file() быстрее, чем fopen+цикл - ускорение 40%**

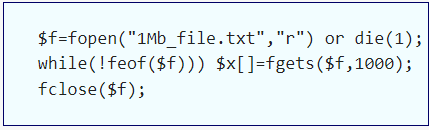
Чтобы прочитать в массив $x файл размером 1Мб (100 000 строк по 10 байт) можно воспользоваться двумя вариантами: чтение файла с помощью file(), либо традиционным методом fopen/fgets. Разумеется, для файлов разного объема и содержимого скорость может меняться. Но в данном примере статистика такова: file("1Mb\_file.txt") работает на 40%быстрее, чем:



Аналогичные варианты



или



работают еще медленнее (во втором случае лишняя функция feof() заметно снижает скорость). Тот же тест, но на 15Мб файле (100 000 строк по 150 байт) показывает разницу в 50%, в пользу file(). Тест проводился так, чтобы исключить фоновый свопинг во время работы из-за предшествующих команд создания/чтения таких больших файлов. Подсчитать тоже самое на очень маленьких файлах в 1-2 Кб не представляется возможным, т.к. операцию чтения нельзя повторять в течении одного теста, операции чтения будут кэшироваться.